# Using the dplace Tool for Pinning

The **dplace** tool binds processes/threads to specific processor cores to improve your code performance. For an introduction to pinning at NAS, see Process/Thread Pinning Overview.

Once pinned, the processes/threads do not migrate. This can improve the performance of your code by increasing the percentage of local memory accesses.

**dplace** invokes a kernel module to create a job placement container consisting of all (or a subset of) the CPUs of the cpuset. In the current **dplace** version 2, an LD_PRELOAD library (libdplace.so) is used to intercept calls to the functions **fork()**, **exec(**), and **pthread_create()** to place tasks that are being created. Note that tasks created internal to glib are not intercepted by the preload library. These tasks will *not* be placed. If no placement file is being used, then the **dplace** process is placed in the job placement container and (by default) is bound to the first CPU of the cpuset associated with the container.

## Syntax

```
dplace [-e] [-c cpu_numbers] [-s skip_count] [-n process_name] \
           [-x skip_mask] [-r [l|b|t]] [-o log_file] [-v 1|2] \
           command [command-args]
dplace [-p placement_file] [-o log_file] command [mpiexec -np4 a.out]
dplace [-q] [-qq] [-qqq]
```

As illustrated above, **dplace** "execs" command (in this case, without its **mpiexec** arguments), which executes within this placement container and continues to be bound to the first CPU of the container. As the command forks child processes, they inherit the container and are bound to the next available CPU of the container.

If a placement file is being used, then the **dplace** process is not placed at the time the job placement container is created. Instead, placement occurs as processes are forked and executed.

## Options for dplace

Explanations for some of the options are provided below. For additional information, see **man dplace**.

### -e and -c *cpu_numbers*

**-e** determines exact placement. As processes are created, they are bound to CPUs in the exact order specified in the CPU list. CPU numbers may appear multiple times in the list.

A CPU value of "x" indicates that binding should *not* be done for that process. If the end of the list is reached, binding starts over again at the beginning of the list.

**-c *cpu_numbers*** specifies a list of CPUs, optionally strided CPU ranges, or a striding pattern. For example:

- -c 1
- -c 2-4 (equivalent to -c 2,3,4)
- -c 12-8 (equivalent to -c 12,11,10,9,8)

- -c 1,4-8,3
- -c 2-8:3 (equivalent to -c 2,5,8)
- -c CS
- -c BT

Note: CPU numbers are *not* physical CPU numbers. They are logical CPU numbers that are relative to the CPUs that are in the allowed set, as specified by the current cpuset.

A CPU value of "x" (or **`*`**), in the argument list for the **`-c`** option, indicates that binding should not be done for that process. The value "**`x`**" should be used only if the **`-e`** option is also used.

Note that CPU numbers start at 0.

For striding patterns, any subset of the characters (**`B`**)lade, (**`S`**)ocket, (**`C`**)ore, (**`T`**)hread may be used; their ordering specifies the nesting of the iteration. For example, **`SC`** means to iterate all the cores in a socket before moving to the next CPU socket, while **`CB`** means to pin to the first core of each blade, then the second core of every blade, and so on.

For best results, use the **`-e`** option when using stride patterns. If the **`-c`** option is not specified, all CPUs of the current cpuset are available. The command itself (which is "execed" by **`dplace`**) is the first process to be placed by the **`-c`** ***`cpu_numbers`***.

Without the **`-e`** option, the order of numbers for the **`-c`** option is not important.

**`-x`** ***`skip_mask`***

Provides the ability to skip placement of processes. The ***`skip_mask`*** argument is a bitmask. If bit *N* of ***`skip_mask`*** is set, then the *N*+1th process that is forked is not placed. For example, setting the mask to 6 prevents the second and third processes from being placed. The first process (the process named by the command) will be assigned to the first CPU. The second and third processes are not placed. The fourth process is assigned to the second CPU, and so on. This option is useful for certain classes of threaded applications that spawn a few helper processes that typically do not use much CPU time.

**`-s`** ***`skip_count`***

Skips the first ***`skip_count`*** processes before starting to place processes onto CPUs. This option is useful if the first ***`skip_count`*** processes are "shepherd" processes used only for launching the application. If ***`skip_count`*** is not specified, a default value of 0 is used.

**`-q`**

Lists the global count of the number of active processes that have been placed (by **`dplace`**) on each CPU in the current cpuset. Note that CPU numbers are logical CPU numbers within the cpuset, not physical CPU numbers. If specified twice, lists the current **`dplace`** jobs that are running. If specified three times, lists the current **`dplace`** jobs and the tasks that are in each job.

**`-o`** ***`log_file`***

Writes a trace file to ***`log_file`*** that describes the placement actions that were made for each fork, exec, etc. Each line contains a time-stamp, process id:thread number, CPU that task was executing on, taskname and placement action. Works with version 2 only.

## Examples of dplace Usage

## For OpenMP Codes

```
#PBS -lselect=1:ncpus=8

#With Intel compiler versions 10.1.015 and later,
#you need to set KMP_AFFINITY to disabled
#to avoid the interference between dplace and
```

```
#Intel's thread affinity interface.

setenv KMP_AFFINITY disabled

#The -x2 option provides a skip map of 010 (binary 2) to
#specify that the 2nd thread should not be bound. This is
#because under the new kernels, the master thread (first thread)
#will fork off one monitor thread (2nd thread) which does
#not need to be pinned.

#On Pleiades, if the number of threads is less than
#the number of cores, choose how you want
#to place the threads carefully. For example,
#the following placement is good on Harpertown
#but not good on other Pleiades processor types:

dplace -x2 -c 2,1,4,5 ./a.out
```

To check the thread placement, you can add the **`-o`** option to create a log:

```
dplace -x2 -c 2,1,4,5 -o log_file ./a.out
```

Or use the following command on the running host while the job is still running:

```
ps -C a.out -L -opsr,comm,time,pid,ppid,lwp > placement.out
```

## Sample Output of log_file

```
timestamp       process:thread cpu taskname| placement action
15:32:42.196786 31044          1 dplace    | exec ./openmp1, ncpu 1
15:32:42.210628 31044:0        1 a.out     | load, cpu 1
15:32:42.211785 31044:0        1 a.out     | pthread_create thread_number 1, ncpu -1
15:32:42.211850 31044:1        - a.out     | new_thread
15:32:42.212223 31044:0        1 a.out     | pthread_create thread_number 2, ncpu 2
15:32:42.212298 31044:2        2 a.out     | new_thread
15:32:42.212630 31044:0        1 a.out     | pthread_create thread_number 3, ncpu 4
15:32:42.212717 31044:3        4 a.out     | new_thread
15:32:42.213082 31044:0        1 a.out     | pthread_create thread_number 4, ncpu 5
15:32:42.213167 31044:4        5 a.out     | new_thread
15:32:54.709509 31044:0        1 a.out     | exit
```

## Sample Output of placement.out

```
PSR COMMAND             TIME   PID  PPID   LWP
  1 a.out           00:00:02 31044 31039 31044
  0 a.out           00:00:00 31044 31039 31046
  2 a.out           00:00:02 31044 31039 31047
  4 a.out           00:00:01 31044 31039 31048
  5 a.out           00:00:01 31044 31039 31049
```

Note: Intel OpenMP jobs use an extra thread that is unknown to the user and it does not need to be placed. In the above example, this extra thread (31046) is running on core number 0.

## For MPI Codes Built with HPE's MPT Library

With HPE's MPT, only 1 shepherd process is created for the entire pool of MPI processes, and the proper way of pinning using **`dplace`** is to skip the shepherd process.

Here is an example for Endeavour:

```
#PBS -l ncpus=8
```

```
....
 mpiexec -np 8 dplace -s1 -c 0-7 ./a.out
```

On Pleiades, if the number of processes in each node is less than the number of cores in that node, choose how you want to place the processes carefully. For example, the following placement works well on Harpertown nodes, but not on other Pleiades processor types:

```
#PBS -l select=2:ncpus=8:mpiprocs=4 ... mpiexec -np 8 dplace -s1 -c 2,4,1,5 ./a.out
```

To check the placement, you can set MPI_DSM_VERBOSE, which prints the placement in the PBS stderr file:

```
#PBS -l select=2:ncpus=8:mpiprocs=4
...
setenv MPI_DSM_VERBOSE
mpiexec -np 8 dplace -s1 -c 2,4,1,5 ./a.out
```

## Output in PBS stderr File

```
MPI: DSM information
grank   lrank   pinning  node name      cpuid
    0       0   yes      r75i2n13           1
    1       1   yes      r75i2n13           2
    2       2   yes      r75i2n13           4
    3       3   yes      r75i2n13           5
    4       0   yes      r87i2n6            1
    5       1   yes      r87i2n6            2
    6       2   yes      r87i2n6            4
    7       3   yes      r87i2n6            5
```

If you use the **`-o`** ***`log_file`*** flag of **`dplace`**, the CPUs where the processes/threads are placed will be printed, but the node names are not printed.

```
#PBS -l select=2:ncpus=8:mpiprocs=4
....
mpiexec -np 8 dplace -s1 -c 2,4,1,5 -o log_file ./a.out
```

## Output in log_file

```
timestamp       process:thread cpu taskname | placement action
15:16:35.848646 19807          - dplace     | exec ./new_pi, ncpu -1
15:16:35.877584 19807:0        - a.out      | load, cpu -1
15:16:35.878256 19807:0        - a.out      | fork -> pid 19810, ncpu 1
15:16:35.879496 19807:0        - a.out      | fork -> pid 19811, ncpu 2
15:16:35.880053 22665:0        - a.out      | fork -> pid 22672, ncpu 2
15:16:35.880628 19807:0        - a.out      | fork -> pid 19812, ncpu 4
15:16:35.881283 22665:0        - a.out      | fork -> pid 22673, ncpu 4
15:16:35.882536 22665:0        - a.out      | fork -> pid 22674, ncpu 5
15:16:35.881960 19807:0        - a.out      | fork -> pid 19813, ncpu 5
15:16:57.258113 19810:0        1 a.out      | exit
15:16:57.258116 19813:0        5 a.out      | exit
15:16:57.258215 19811:0        2 a.out      | exit
15:16:57.258272 19812:0        4 a.out      | exit
15:16:57.260458 22672:0        2 a.out      | exit
15:16:57.260601 22673:0        4 a.out      | exit
15:16:57.260680 22674:0        5 a.out      | exit
15:16:57.260675 22671:0        1 a.out      | exit
```

## For MPI Codes Built with MVAPICH2 Library

With MVAPICH2, 1 shepherd process is created for each MPI process. You can use **`ps -L -u`** ***`your_userid`*** on the running node to see these processes. To properly pin MPI processes using **`dplace`**, you cannot skip the shepherd processes and must use the following:

```
mpiexec -np 4 dplace -c2,4,1,5 ./a.out
```

---

Article ID: 284
Last updated: 23 Dec, 2020
Revision: 65
Porting/Building Code -> Optimizing/Troubleshooting -> Process/Thread Pinning -> Using the dplace Tool for Pinning
https://www.nas.nasa.gov/hecc/support/kb/entry/284/